Manfred M. Fischer

Computational Neural Networks: A New Paradigm for Spatial Analysis

Paper

WSG Diskussionspapiere

WSG 54/96

Computational Neural Networks:

A New Paradigm for Spatial Analysis

Manfred M. Fischer

Institut für Wirtschafts- und Sozialgeographie

Wirtschaftsuniversität Wien

Department of Economic and Social Geography

Vienna University of Economics and Business Administration

WSG Discussion Papers

Abteilung für Theoretische und Angewandte Wirtschafts- und Sozialgeographie
Institut für Wirtschafts- und Sozialgeographie
Wirtschaftsuniversität Wien

Vorstand: o.Univ.Prof. Dr. Manfred M. Fischer
A - 1090 Wien, Augasse 2-6, Tel. (0222) 313 36 - 4836

Redaktion: Mag. Petra Staufer

WSG 54/96

Computational Neural Networks:
A New Paradigm for Spatial Analysis

Manfred M. Fischer

WSG-Discussion Paper 54

August 1996

ISBN 3 85037 063 1

1. Introduction

Spatial analysis (SA) represents one of the currently most valuable geographical assets that has a number of practical applications outside of geography. It is the spatial perspective that distinguishes spatial analysis from other forms of data analysis. There are at least three arguments why a spatial perspective in data analysis might be important (Goodchild et al. 1992). *First*, space provides a simple, but very useful framework for handling large amounts of data. *Second*, the spatial perspective permits easy access to information on the relative location of objects and events. *Third*, the distance between objects and events is often an important factor in interaction between them, both in environmental and socio-economic applications.

Spatial analysis, as it has become over the past decades, basically includes two major fields: spatial data analysis and spatial modelling though the boundary is rather blurred. In order to focus the discussion, we will essentially limit the scope to the first. Spatial data analysis (SDA) may be defined as technology (i.e. body of methods and techniques) for analyzing events (objects) where the results of analysis depend on the spatial arrangement of the events (see Haining 1994). Hereby events may be represented in form of point, line or area objects in the sense of spatial primitives, located in geographical space, attached to which is a set of - one or more - attributes. Location, topology, spatial arrangement, distance and spatial interaction have become a major focus of attention in activities dealing with detecting patterns in spatial data, exploring and modelling relationships between such patterns.

Typically, two different types of information are characteristic for SDA: locational (geometric/topological) information about the spatial objects of concern; and attribute information about these. It is the first type of information which complicates SDA because location leads to effects which render classical statistical techniques unsafe, i.e.

- *first*, to spatial dependence which directly results from Tobler's (1979) 'First Law of Geography' where 'everything is related to everything else, but near things are more related than distant things', and
- *second*, to spatial heterogeneity or non-stationarity related to spatial differentiation which follows from intrinsic uniqueness of each location (Anselin 1994).

The complications are similar to those found in time series analysis, but are exacerbated by the multi-directional, two-dimensional nature of dependence in space (Griffith 1993).

Most of the SDA techniques and methods currently in use were developed in the 1960s and 1970s, i.e. in an era of scarce computing power and small data sets. Their current implementations take only limited advantage of the data storage and retrieval capabilities of modern computational techniques, and basically ignore both the emerging new era of parallel supercomputing and the computational intelligence techniques. In addition, they overemphasize linear statistical model designs while non-linearities prevail in reality, tend to neglect rather than take into account the special nature of spatial data, and exhibit major difficulties to cope with the information rich worlds on which societies and economies increasingly depend.

No doubt, SDA is currently entering a period of rapide change, a period which presents the unique opportunity for new styles of data analysis in order to meet the new needs for efficiently and comprehensively exploring large databases for patterns and relationships against a background of data uncertainty and noise, especially when the underlying database is of the order of gigabytes. It is argued in this paper that computational intelligence technologies in general and computational neural networks in particular show the potential for a new paradigm in spatial data analysis providing

analysts with rich and interesting classes of novel data driven methods and techniques applicable to a wide range of domains in spatial analysis.

The paper is structured as follows. In section 2, we attempt to clarify what we mean with computational intelligence in contrast to artificial intelligence. Computational intelligence is currently best designed in capturing those systems which exhibit learning as a fundamental property. Learning in such systems can be understood as a change in behaviour brought about by experience. Thus, the focus of this paper is on computational neural networks (CNN). In such networks learning takes the form of approximating relationships from data, or the form of encoding desired equilibria. In section 3 we consider some fundamental characteristics of these computational tools, while feedforward neural networks which provide spatial analysts with novel and extremely useful classes of mathematical tools are described in some detail in section 4. Section 5 deals with supervised training of such networks and briefly reviews a variety of powerful local and global optimization techniques. The processing demands of large-size real world applications may be prohibitive for developing application domain specific fully automatic CNN-systems. Thus, section 6 considers various parallelization techniques, software and hardware related approaches, to reduce processing time. Some conclusions are provided in the final section.

2. What is Computational Intelligence?

Attempts to artificially mimic intelligence have a large history. Langton (1989) traces the history of artificial life from the pneumatical animal gadgets of Hero of Alexandria in the first century, via Johann von Neumann's first computational approach to machine reproduction behaviour to Norbert Wiener's cybernetics. More recently, the field of

Arificial Intelligence (AI) has attempted to capture the essence of intelligence. What is Computational Intelligence (CI) and how does it differ from AI?

Computational intelligence is a collective term, comprising emergent technologies such as artificial life, evolutionary computation, neural networks and their likes, which was introduced by James C. Bezdek at the 1994 IEEE World Congress on Intelligent Systems held in Orlando, and has since been adopted as an official IEEE standard. It denotes the lowest-level forms of 'intelligence' which stem from the ability to process numerical (low-level) data without using knowledge in the AI sense. In addition, a computationally intelligent system begins to exhibit computational adaptivity, fault tolerance, speed approaching human-like turnaround, and error rates which approximate human performance (see Bezdek 1994).

An artificially intelligent system is a CI system where added value comes from incorporating knowledge in form of non-numerical information, rules and constraints that humans process. Thus, neural networks such as feedforward pattern classifiers, self-organizing maps, learning vector quantization, adaptive resonance theories, etc. are generally CI rather than AI systems. Ignoring the distinction between artificial and computational intelligence may lead to confusion, misunderstanding, misrepresentation and misuse of neural network models in spatial analysis.

Much of the recent interest of geographers in neural network modelling (see, e.g., Openshaw 1993b, Leung 1996) stems from the growing realization of the limitations of conventional SDA tools as vehicles for exploring patterns and relationships in geographic information systems and remote sensing environments, and from the consequent hope that these limitations may be overcome by judicious use of computational neural networks.

3. Computational Neural Networks

Computational neural networks are parallel distributed information processing structures consisting of simple, but generally non-linear processing (computational) elements which can possess a local memory and can carry out localized information processing operations with adaptation capabilities, massively interconnected via unidirectional signal conduction paths called connections. Each connection has a weight associated with it that specifies the strength of this link. A processing element (PE) can receive any number of incoming connections and has a single output connection which can branch into copies to form multiple output connections, where each carrying the same signal. The information processing active within each PE can be defined arbitrarily with the restriction that it has to be completely local, i.e. it has to depend only on the current values of the input signals arriving at the PE and on values stored in the PE's local memory (see Hecht-Nielsen 1990).

Figure 1: A typical neural network architecture

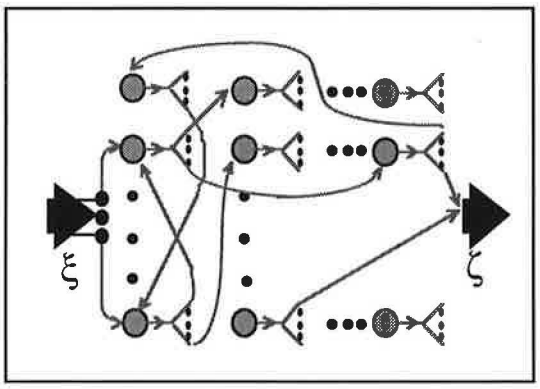


Figure 1 shows a typical neural network architecture. The input to the network considered as a data array ξ and the output of the network as a data array ζ. Viewed in this manner the general functional form of a network is similar to that of a software procedure 'input → processing → output'. Whether implemented in parallel hardware or

simulated on a von Neumann computer, all computational networks consist of a collection of simple computational elements that work together to solve problems.

Figure 2 reflects current thinking about information processing that should be performed at each PE in a computational neural network. Characteristically, two mathematical functions are active at each PE. The first mathematical function is an *integrator funtion*, say f , which integrates the connection weights, say $\mathbf{w}=\{w_i\}$, with the input signals, say $\xi=\{\xi_i\}$, arriving via the incoming connections which impinge upon the processing element. The first entry in each vetor in Figure 2 is shown by a dotted line to indicate the bias weight w_0 connected to a constant input $\xi_0=1$. Typically f is theinner product, usually the Euclidean dot product, say

$$\eta = f(\xi) = <\xi, \mathbf{w}> = \sum_{i=1,\ldots,k} \xi_i \mathbf{w}_i + \mathbf{w}_0 \qquad (1)$$

where $\mathbf{w}_0$ is an unknown parameter which has to be predefined or learned during training. $\mathbf{w}_0$ represents the offset from the origin of $\Re^k$ to the hyperplane normal to $\mathbf{w}$ defined by f. Without loss of generality the augmented vectors $\xi=(1, \xi_1,\ldots, \xi_k)^T$ and $\mathbf{w}=(\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_k)^T$ may be considered as input and weight vectors, respectively, in $\Re^{k+1}$. A processing element with this type of integrator function is called *first-order processing* element because f is an affine function of its input vector ξ. When the inner product f is replaced by a more complicated function, higher-order processing elements arise. For example, a second order processing element may be realised with a quadratic form, say $\xi^T\mathbf{w}\xi$, in ξ. It is important to note that each processing element may be viewed as having (k+1) unknowns, but only k inputs.

Each processing element typically applies a transfer (or activation) function, say F, to the value of the integrator function (or activation) on its inputs. The transfer function

produces the processing element's output signals. The transfer functions in many fundamental CNNs satisfy

$$F(\eta) \rightarrow \begin{cases} 1 & \text{as } \xi \rightarrow +\infty \\ 0 & \text{as } \xi \rightarrow -\infty \end{cases} \tag{2}$$

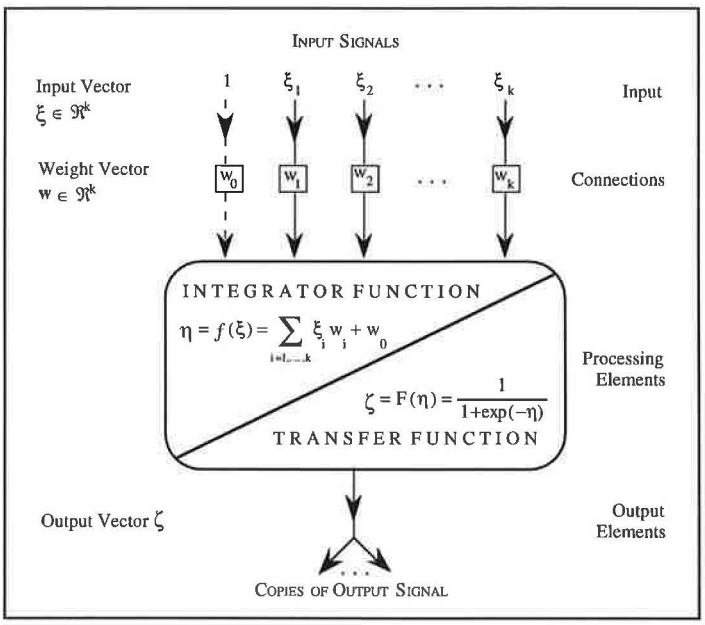
which are called sigmoid functions. A common value choice in the case of continuous inputs is the logistic function shown in figure 2.

A third mathematical operation for a computational neural network is the *update function*

$$\mathbf{W}(t+1) = U(\mathbf{W}(t)) \tag{3}$$

where $\mathbf{W}(t) = (\mathbf{w}_1(t), ..., \mathbf{w}_N(t))$ denotes the network weight vector, i.e. the collection of all the individual vectors at the N PEs in the network at any time (iteration) t. The weight vector $\mathbf{w}_n(n=1,...,N)$ is stored in the n-th processing element's local memory. Updating is done during training.

Figure 2: Information processing at the processing element



Although a vast variety of NN models exist, and more continue to appear as research continues, many of them have common topological characteristics, properties of the PEs, and training [learning] approaches (see, e.g., Hecht-Nielsen 1990, Carpenter and Grossberg 1991, Kosko 1992, Wasserman 1993). Basically three entities characterize a computational NN (see Fischer and Gopal 1993):

- the network topology or interconnection of its PEs [called *architecture*],
- the characteristics of its PEs, and
- the method of determining the weights at the connections [called *training* or *learning strategy*].

Different interconnection strategies lead to different types of NN architectures (for example, feedforward versus recurrent) which require different learning (training) strategies. At the most fundamental level two categories of training may be distinguished: supervised and unsupervised. In *supervised learning* the network is trained on a training set consisting of a sequence of input and target output data. Training is accomplished by adjusting the network weights so as to minimize the difference between the desired and actual network outputs. *Unsupervised learning* (also called self-organization) requires only input data to train the network. During the training process the network weights are adjusted so that similar inputs produce similar outputs. This is accomplished by a training algorithm that extracts statistical regularities from the training set, representing them as the values of network weights (see Fischer and Gopal 1994b, Fischer 1995).

4. Feedforward CNNs - an Attractive Class of Mathematical Tools

Multilayer feedforward computational neural networks such as perceptrons and radial basis function networks have recently emerged as attractive class of CNNs based upon

sound theoretical concepts. They map $\xi_1,...,\xi_I$ inputs into $\zeta_1,...,\zeta_P$ outputs, and may be viewed as generalized non-linear extensions of conventional spatial statistical models such as, e.g., spatial regression models, spatial interaction models, and linear discriminant functions. To better understand this relationship, we explicitly express feedforward CNNs mathematically, and consider for this purpose feedforward CNNs with inputs $\xi_1,...,\xi_I$, one hidden layer of j=1,..., J computational units and - for simplicity's sake - one output unit ζ, only. Such networks may be mathematically expressed as

$$\zeta = g_2\Big(\sum_{j=0,...,J} \mathbf{w}_j^{(2)}\ g_1\Big(\sum_{i=0,...,I} \mathbf{w}_{ij}^{(1)}\ \xi_i\Big)\Big) = g(\xi, \theta) \qquad (4)$$

where the parameters $\mathbf{w}_{ij}^{(1)}$(i=1,...,I; j=1,...,J) denote weights associated with connections from the input array of I units to the hidden layer, and the parameters $w_j^{(2)}$ (j=1,...,J) those weights associated with connections from the hidden layer to the output unit. The bias have been absorbed into the weights. g_1 and g_2 represent transfer functions of the PEs at the hidden and output layer, respectively. The expression $g(\xi,\theta)$ is a convenient short-hand for network output since this depends only on inputs and weights. The symbol ξ represents a vector (list) of all the input values, and the symbol θ represents a vector of all the weights (the $\mathbf{w}_{ij}^{(1)}$s and $\mathbf{w}_j^{(2)}$s). g might be called the network output function. The transfer functions of the hidden and output unit determine the precise form of the function g.

Different types of transfer functions g_1 and g_2 will lead to different particular computational networks. If the transfer functions are taken to be linear so that $g_1(a)=g_2(a)=a$, functional form (4) becomes a special case of the general linear regression model. The crucial difference is that here we consider the weight parameters appearing in the hidden and output layers as being adaptive so that their values can be

changed during the process of network training (in statistical terminology: parameter estimation).

The novelty and fundamental contribution of the feedforward neural network approach to spatial analysis derives from its focus on functions such as (4), and much less on the associated learning methods which will be discusseed in section 5. Among others Hornik et al. (1989) have demonstrated that the network output function g can provide an accurate approximation to any function of ξ likely to be encountered, provided that the number J of hidden units is sufficiently large. Because of this universal approximation property, one hidden layer feedforward networks are useful for applications in pattern recognition and classification, discrimination, regression, forecasting and a host of related spatial analysis tasks.

Feedforward CNN modelling as universal function approximators may be considered as a *three-stage process* as outlined in Fischer and Gopal (1994a):

- The *first stage* refers to the identification of a model candidate from a family of two-layer feedforward networks with specific types of non-linear processing elements.
- The *second stage* involves the estimation of the network parameters of the selected neural network model and the optimization of the model complexity for the given training set.
- The *third stage* is concerned with testing and evaluating the out-of-sample [generalization] performance of the model.

One critical issue for a successful application of CNNs to spatial analysis is the complex relationship between learning (training) and generalization. It is important to stress that the *ultimate goal of network training* is not to learn an exact representation of the training data itself, but rather to build a model of the process which generates the

data in order to achieve a good generalization [out-of-sample] performance of the model. One way to optimizing the generalization performance of a model is to control its effective complexity where complexity being measured in terms of network parameters. This problem of finding the optimal complexity for a neural network model - though crucial for a successful application - has been highly neglected in applications up to now. In principle, there are three classes of techniques to control overfitting of a model:

- *regularization techniques* [i.e. add a penalty term to the error function],
- *network pruning techniques* [i.e. train an overly-large network and successively delete weights, as illustrated, e.g., in Fischer et al. 1994], and
- *cross-validation techniques* to determine when to stop training [i.e. early stopping heuristic, as illustrated, e.g., in Fischer and Gopal 1994a].

The point of best generalization is determined by the trade-off between the bias and the variance of the model, and occurs when the combination of bias and variance is minimized. In the case of feedforward networks it is possible - by using a sequence of successively larger data sets, and a corresponding set of models with successively greater complexity - to reduce both bias and variance simultaneously and, thus, to improve the generalization performance of the model. The generalization performance which might be achieved is, however, limited by the intrinsic noise of the data.

The feasibility of computational feedforward networks for spatial analysis tasks has been demonstrated, first, in the context of spatial interaction modelling with noisy real world data of limited record length to model interregional telecommunication traffic in Austria (Gopal and Fischer 1993, 1996, Fischer and Gopal 1994, Leung et al. 1996) or using UK journey-to-work flows (Openshaw 1993b), and, second, in the context of urban land cover from satellite imagery with noisy real data of larger record length (see

Fischer et al. 1994 among others, and Wilkinson 1996 for an overview). The attractivity of this novel approach essentially stems from the following features of CNNs:

- the greater representational flexibility and freedom from linear model design constraints as illustrated by (4),
- the built-in capability [via net representation, training] to incorporate rather than ignore the special nature of spatial data;
- the greater degree of robustness or fault tolerance to deal with noisy data, missing and fuzzy information;
- the ability to deal efficiently with very large spatial data sets, and thus the prospect to obtain better results by being able to process finer resolution data or real-time analysis;
- the built-in capability to dynamically adapt the connection weights to changes in the surrounding environment [learning]; and
- generalization [out-of-sample performance] capabilities in a very specific and generally satisfying sense.

5. Training of Feedforward Neural Networks - A Variety of Optimization Techniques

The essence of network learning is to find a suitable set of parameters that approximate an unknown input-output relation of type (4). This problem is generally solved using supervised learning techniques. Supervised learning requires a training set (i.e., a set of input-target output examples). Learning the training set may be posed as a search in the network parameter space by introducing a performance (error) measure, i.e. a function of the adaptive network parameters, that measures the quality of the network's approximation to the input-output relation on the restricted domain covered by the training set. The minimization of this error over the network's parameter space is called

the training process. The task of learning, however, is to minimize that error for all possible examples related through the input-output relation, namely, to generalize outside of the training set.

A frequently encountered performance measure is the squared error (for more details see, e.g., Gopal and Fischer 1996a)

$$E(\zeta^* \mid \xi, \theta) = \frac{1}{2}(\zeta^* - g(\xi, \theta))^2 \qquad (5)$$

for a training set available consisting of a vector (list) ξ of all the input training patterns together with observations on corresponding target variables ζ^*.

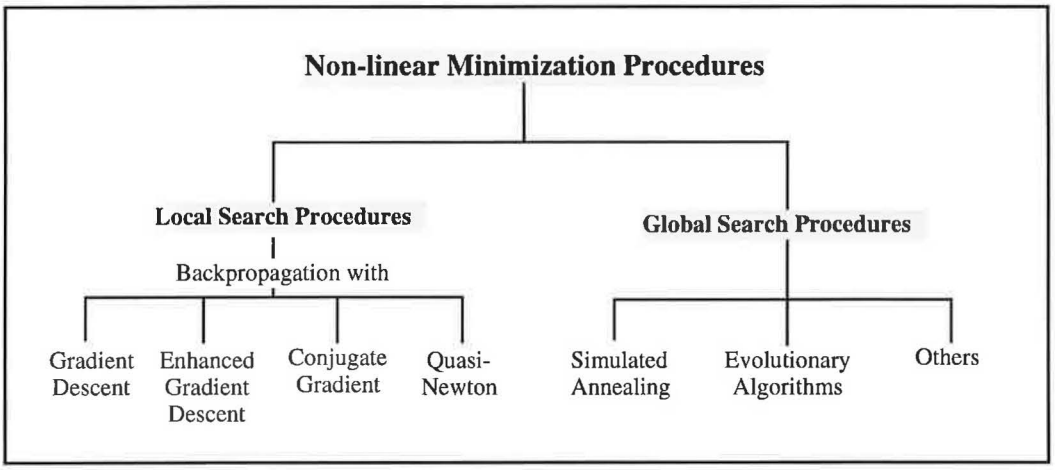
CNN training strives to minimize the chosen error function such as (5). Due to the non-linearity of the transfer functions g_1 and g_2, it is not possible to find closed-form solutions for this optimization problem. But there is a considerable variety of local and global search procedures available (see figure 3). Minimization techniques are termed local if the computations needed to update each network weight (see equation (3)) can be performed using local information to that weight only. This may be motivated by the desire to implement network training algorithms in parallel hardware.

Local training techniques generally involve an iterative procedure to minimize a performance measure such as (5), with adjustments to the network parameters θ being made in a sequence of iteration steps. At each iteration stage we can distinguish between two different stages. In the *first stage*, the derivatives of the performance function with respect to the network parameters have to be evaluated. This evaluation is most commonly performed by the *backpropagation technique* which provides a computationally efficient procedure for evaluating such derivatives (Rummelhart et al. 1986). At this stage errors are propagated backwards through the network to the output

processing units. It is important to note that the backpropagation technique can also be used for the evaluation of other derivatives such as the Jacobian and Hessian matrices. In the second stage, the derivatives are utilized to compute the parameters adjustments. For this *stage of weight adjustment* a wide range of optimization procedures may be used (Bishop 1995).

The simplest and most popular of such optimization procedures is the *gradient* (also known as steepest) *descent*. Gradient descent techniques involve taking a sequence of iteration steps through the parameter space. With the simple gradient descent the direction of each step is given by the local negative gradient of the performance function chosen, while the step size performed is determined by an arbitrary parameter, called learning rate.

Figure 3: A taxonomy of CNN training procedures



In the pattern-based (also termed on-line) version of the gradient descent, the error function gradient is evaluated for just one training pattern at a time and the parameter values updated where the different patterns in the training set may be used either in sequence (deterministic pattern-based version) or selected at random (stochastic pattern-based version). The stochastic version shows the potential advantage to escape

from local minima. The pattern-based versions tend to be superior to the batch version especially for large and redundant training sets (Hertz et al. 1991). The training of CNNs using the backpropagation in combination with the basic gradient descent is plagued by slow convergence in the case of larger training sets. Numerous heuristic optimization algorithms have been proposed to improve the convergence speed of the gradient descent technique. Examples include gradient descent with momentum update, the delta-delta rule, the delta-bar-delta rule (see Jacobs 1988) and a heuristic scheme known as quickprop (Fahlman 1988), to mention just the most popular ones.

Another important class for weight adjustment is based on the concept of *conjugate gradients*. Conjugate gradient procedures provide minimization techniques which require only the evaluation of the error function and its derivation, and utilize information about the direction search from the previous iteration in order to accelerate the convergence. Each search direction is conjugate if the performance function is quadratic. Theoretically, this procedure guarantees to minimize a quadratic error function in q or fewer iterations (batch mode) where q is the dimensionality of the parameter space. It is interesting to note that the conjugate gradient procedure may be regarded as a form of gradient descent with momentum, where the learning rate is determined by line search.

Quasi-Newton (also called variable-metric) procedures, the third class of local search techniques, are today the most efficient and sophisticated optimization algorithms, including the Davidson-Fletcher-Powell (DFP) and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) techniques. They iteratively use successively improved approximations to the inverse Hessian instead of the true inverse as in the basic Newton procedure, and utilize only information on the first derivatives of the performance function. Quasi-Newton procedures require much more storage, but only half of the gradient evaluations in comparison to conjugate gradient descents. In cases of CNNs with more

than a few thousand network parameters procedures such as conjugate gradients have significant advantage over the quasi-Newton techniques (Shanno 1990).

Local minimization algorithms find the local minima efficiently and work best in unimodal problems. They show difficulties when the surface is flat (i.e. gradients close to zero), when gradients can be in a large range, or when the surface is very rugged. To overcome local search deficiencies, global minimization procedures may be used. Two approaches are worthwhile to mention here: simulated annealing and genetic search algorithms.

Simulated annealing is a stochastic global optimization procedure based on a strong analogy between the physical annealing process of solids and the problem of solving large combinatorial optimization problems (Kirkpatrick et al. 1983, Press et al. 1992). The basic idea is to start at some initial point in parameter space, take a step, and evaluate the performance function once. When minimizing a function like (5), it accepts any downhill movement and repeats the process from this new starting point. It allows moves uphill to escape from local minima in a controlled fashion, so that there is no danger of jumping out of a local minimum and falling into a worse one. As the minimization process proceeds, the step length is lowered at an appropriate rate so as to control the probability of jumping away from relatively good minima. The search converges to a local - sometimes global - minimum. Hajek (1988) gives a useful survey and some theorems establishing conditions under which simulated annealing may lead to a global optimum. The efficiency of the simulated annealing approach depends on the choice of the schedule for a control parameter. The major drawback of the procedure is a very long computation time since it is necessary to perform a large number of random searches at each step to arrive near optimal solutions.

Genetic algorithms are probabilistic and highly parallel mathematical algorithms that transform a set (population) of individual objects (characteristically fixed-length character strings patterned after chromosome strings), each with an associated fitness value, into a new population (i.e., the next generation) using operations inspired by the Darwinian principles of reproduction and survival of the fittest and applying genetic operations such as reproduction, crossover (sexual recombination) and mutation (Koza 1992). There are four major steps in preparing to use the conventional genetic algorithm of fixed-length character strings to solve the minimization problem at hand. These include (Koza 1992, 1994):

- ❒ specification of the representation scheme,
- ❒ specification of the fitness measure,
- ❒ specification of the parameters and variables controlling the algorithm, and
- ❒ specification of the criteria for terminating a run.

The representation scheme is a mapping that represents each possible point in the weight space as a fixed-length character string over some alphabet. Specification of the representation scheme requires then to select the string length and the alphabet size. The fitness measure (in our context the opposite of the performance measure such as (5)) assigns a fitness value to each possible individual (point in the weight space) in the population. The primary parameters for controlling the genetic algorithm are the population size M and the maximum number G of generations to be run (termination criterion). Secondary parameter control the frequencies of reproduction, crossover (distant search in the weight space) and mutation (local search in the weight space). In principle, three steps can be distinguished in executing the conventional genetic algorithm (Koza 1994):

- Randomly create an initial population of M points θ^m (m=1,...,M) in the search space.
- Iteratively execute the following substeps on the population until the termination criterion has been satisfied:
  (i) assign a fitness value to each individual using the chosen fitness measure, and thus evaluate each individual for fitness,
  (ii) create a new population of strings by applying the following three genetic operators to individual strings in the population chosen with a probability based on fitness: *first*, reproduction of an existing individual string by copying it into the new population; *second*, creation of two new strings by genetically recombining substrings using the crossover operation at a randomly chosen crossover point; and *third*, creation of a new string from an existing by randomly mutating the character at one randomly chosen position in the string.
- The best individual string that appeared in any generation (i.e. the best-so-far individual) is viewed as the result of the genetic algorithm for the run and represents the solution obtained by the genetic algorithm to the minimization problem.

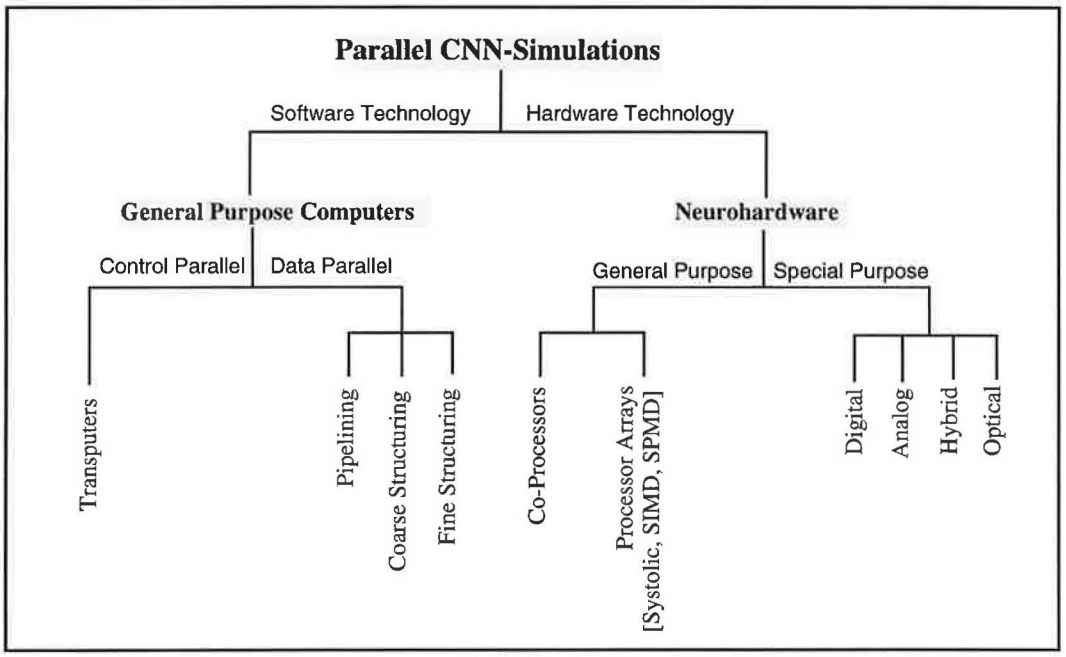
It is important to note that the genetic algorithm - like other iterative search procedures - requires to perform multiple independent runs in order to arrive at a satisfying solution. In practice, the genetic algorithm tends to be surprisingly fast in effectively searching complex, highly non-linear, multidimensional search spaces (see, e.g., Leung et al. 1996). But there is no general theoretical result guaranteeing that indeed a close to optimal solution has been achieved.

6. Automating CNN-Systems: Technical Issues to Reduce Processing Time

CNN tools along with a wide variety of powerful training techniques now exist to solve high dimensional problems of pattern recognition and classification of massive quantities of spatial data, to intelligently allow the user to sift through the data, reduce dimensionality, and find patterns of interest in data rich environments such as GIS and RS. A critical step in the development of CNN-systems for tackling such hard problems is the creation of application domain specific fully automatic systems (e.g. satellite imagery classification systems for detecting land cover) which require no user intervention. Most spatial analysts do not have the expert knowledge to decide which CNN architecture to use, how to set up training data, to organize training and evaluate out-of-sample performance, to control for model complexity, etc. In automatic systems all aspects of architecture selection, parameter setting, performance evaluation and, if necessary, re-training has to be handled automatically. There is no reason why this can not be done. But it requires a new emphasis on CNN-software engineering with end-user requirements in mind (see Wilkinson 1996).

One important requirement in the automatic use of CNNs is the possibility for training to be controlled automatically so that good generalization (i.e. out-of-sample) performance is achieved. Since analytical results on feedforward CNNs do not provide more than very general guidance on the specification of an ideal CNN architecture or learning parameters for a training session, it is necessary to allow an automatic system to experiment with different configurations in a short time frame. While many CNN applications may not need tremendous speed, automatic systems with the ultimate goal of offering fast CNN prototyping against user defined goals demand such performance. Moreover, processing of very large CNNs, with thousands of processing units may only be practical when exploiting recent software and hardware related developments.

Figure 4: A classification of parallelization approaches for CNN-simulation (see Serbedzija 1996)



Since parallelization lies at the very heart of CNNs it seems reasonable to look more closely at the various parallelization approaches available to reduce processing time. At the most fundamental level one may distinguish two distinct approaches (see figure 4): parallel simulation of CNNs on general purpose computers (a software approach), and CNN-simulation/emulation on neurohardware (a hardware approach). These approaches will be briefly characterized in the sequel with a focus on different parallelization techniques.

The *software related approach* is characterized by the search for appropriate parallelization techniques on conventional von Neumann processors. CNN-simulations can be parallelized in several ways. Following Nordstrom and Svensson (1992) and with feedforward CNNs in mind, one can distinguish between structuring approaches that lead to the following levels of parallelism:

- training-session parallelism (i.e. simultaneous execution of different training sessions),
- training-example parallelsim (i.e. simultaneous on different training patterns),
- layer-parallelism (i.e. concurrent execution of layers within a CNN),
- PE-parallelism (i.e. parallel execution of computational units for a single input), and
- weight-parallelism (i.e. simultaneous summation within a PE),

where each approach refines the preceding one in a number of possible parallel activities.

Depending on the presence or absence of central control, parallel computer architectures as possible hosts for CNN-simulations may be divided into two broad classes: *Data parallel* and *control parallel architectures* (see figure 4) that require quite different programming styles. Control parallel architectures perform information processing in a decentralized way. Decentralized control and decentralized data distribution are important features of control parallel programming techniques. A parallel program is explicitly disaggregated into several different tasks that are placed on different processors. Each processor executes a different program. Synchronization is explicit. The communication scheme is usually general routing (Serbedzija 1996). The most popular control parallel host for CNN-simulations is the transputer system which may be viewed as a virtual neurocomputer (virtual PEs, partitioning), i.e. the PEs or connections - not currently using the hardware - sit in a memory structure waiting their turn to use the hardware (Staub et al. 1991).

In contrast, data parallel architectures simultaneously process large data sets. Centralized control and decentralized data distribution are characteristic for data parallel programming techniques. The data are processed by numerous processors in a

synchronous or regular manner that execute the same program. Synchronization is centralized. The communication scheme is broadcast or circulation. Data parallel CNN-simulations exploit all the above mentioned parallelization techniques. Most popular are coarse-structuring (providing training-example and PE per layer parallelization); fine-structuring (providing PE and weight-parallelization); and pipelining (providing layer - and sometimes PE - parallelization) (Serbedzija 1996). The most popular data parallel hosts for CNN-simulations include the Connection Machine (Rosenberg and Blelloch 1987) and systolic arrays (Chung et al. 1992).

Implementation in hardware specifically designed for CNNs could take advantage of the parallel nature of CNNs and run much faster, often by orders of magnitude. *Hardware solutions* to reduce processing time include *general purpose and special purpose neuroarchitectures* (see figure 4). General purpose neuro- hardware is based on generic CNN-features and, thus, supports different CNNs. It comes in many varieties and flavours. At the most fundamental level *processor arrays* and *co-processors* (neuro-accelerators) may be distinguished. The latter are the most popular hardware upgrades for CNN-applications, because of their low price and wide availability. They usually come with software interfaces, drivers and libraries, and are simple boards which can be added to workstations and PCs converting them to neurocomputers. Systems with co-processors are several thousand times faster than standard workstations. Processor arrays are complex VLSI architectures organized in a data parallel manner. Three approaches dominate in practice: systolic arrays, processor arrays of the SIMD type and processor arrays of the SPMD type.

Highly parallel processor arrays offer even better performance than neuro-accelerators, but often lack flexibility and scalability. The Synapse system produced by Siemens is one of the most popular general purpose neurocomputers. It is based on 2D systolic architecture designed to accelerate matrix operations. The standard configuration

consists of eight pipelined MA16 chips, each with its own off-chip weight memory and a throughput of 500 M CPS (connections per second), two MC68040 CICS processors for control purposes, and a 128 MByte DRAM. The full system, connected to a workstation, performs 5.12 G CPS and 33 M CUPS (connection updates per second) (Serbedzija 1996).

The best performance of CNNs in terms of processing time is achieved with special purpose neurocomputers which implement a particular computational network (e.g. feedforward networks) in silicon, using state-of-the-art digital or analog technology. The advantages of digital over analog technology include the use of well understood fabrication techniques, RAM weight storage, flexible design, mathematical accuracy, etc. (Lindsay and Lindblad 1995). Analog neurocomputers are faster and closer to CNN paradigms, but require sophisticated design and fabrication tools. One of the fastest neurochips is Hitachi's Wafer Scale Integration chip [WSI] which has 576 digital processing units and 36K weights integrated onto a 5-inch silicon wafer using 0.8 μm CMOS. The system of eight WSI boards performs 2.3 G CUPS, measured on a feedforward network with backpropagation training (Serbedzija 1996).

Hybrid chips exploit the advantages of both digital and analog techniques. Digital techniques, for example, might be applied to perform accurate and flexible training, while the analog chip's potential density might be utilized to obtain finer parallelization on a smaller area in the recall stage (i.e. the retrieval of the stored weights from the trained CNN to new inputs). A successful hybrid implementation is Mitsubishi's Boltzmann machine with a training speed of 28 G CUPS achieved by digital circuits, and a maximal recall speed of 1 trillion CPS (connections per second).

It seems to be likely that silicon technology will continue to dominate special purpose CNN architecture in the near future. Optical technology introducing photons as basic

information carriers and, thus, being much faster than electrons puts optical neurocomputing first among possible candidates for the neurocomputer in the future. Because of the low price and the wide availability there is no doubt that neuro-accelerators being the first choice for development activities of domain specific automated CNN-systems in the next years to come.

7. Conclusions and Future Work

Spatial analysis is entering a new era of data driven exploratory data searches for patterns and relationships in the context of an analysis process increasingly driven by the availability of huge quantities of spatial data. Computational neural networks provide an interesting and powerful paradigm to meet the new challenges, one that is likely to slowly evolve rather than revolutionize with major radical change over a short time frame. The driving force is a combination of large amounts of spatial data due to the GIS and RS data revolutions, the availability of attractive and novel CNN-tools, the emergence of powerful neurohardware, and the new emphasis on exploratory data analysis and modelling.

Computational neural networks provide not only novel and extremely valuable classes of data-driven mathematical tools as illustrated in this paper with feedforward networks in mind, but also an appropriate framework for re-engineering our well established spatial analysis tools to meet the new large scale data processing needs in data rich environments. The most important challenges in the years to come are, first, to develop application domain specific methodologies relevant for spatial analysis; second, to gain deeper theoretical insights into the complex relationship between training and generalization which is crucial for the success of real world applications; and, third, to deliver high performance computing on neurohardware to enable rapid CNN

prototyping to take place with the ultimate goal to develop application domain specific automatic CNN-systems. This is crucial for making CNNs just another element in the toolbox of spatial analysts.

Acknowledgement: The author gratefully acknowledges funding provided by the Austrian Federal Ministry for Science, Research and Arts (research contract GZ 308.937/2 - IV/3/95).

References

Anselin L 1994 Exploratory spatial data analysis and geographic informations systems, in **EUROSTAT 3D: New Tools for Spatial Analysis**. Luxembourg: 45-54

Bezdek J C 1994 What is computational intelligence, in Zurada J M, Marks II R J, Robinson C J (eds): **Computational intelligence: imitating life**. New York, IEEE: 1-12

Bishop C M 1995 **Neural networks for pattern recognition**. Clarendon Press, Oxford

Carpenter G A, Grossberg S (eds) 1991 **Pattern recognition by self-organizing networks**. Cambridge [MA], The MIT Press

Chung J-H, Yoon H, Maeng S R 1992 A systolic array exploiting the inherent parallelism of artificial neural networks, **Microprocessing and Microprogramming**, 33(3): 145-59

Davis L (ed) 1991: **Handbook of genetic algorithms**. Van Nostrand Reinhold, New York

Fahlman S E 1988 Faster-learning variations on back-propagation: An empirical study, in Touretzki D, Hinton G E, Sejnowski T J (eds): **Proceedings of the 1988 Connectionist Models Summer School**, 38-51. Morgan Kaufmann, San Mateo (CA)

Fischer M M 1995 Fundamentals in neurocomputing. In Fischer M M, Sikos T T, Bassa L (eds) **Recent developments in spatial information, modelling and processing**. Budapest, Geomarket Co: 31-41

Fischer M M, Gopal S 1994a Artificial neural networks. A new approach to modelling interregional telecommunication flows. **Journal of Regional Science** 34 (4): 503-27

Fischer M M, Gopal S 1994b Neurocomputing and spatial information processing. From general considerations to a low dimensional real world application. In **EUROSTAT 3D: New Tools for Spatial Analysis**. Luxembourg: 55-68

Fischer M M, Gopal S 1993 Neurocomputing - a new paradigm for geographic information processing. **Environment and Planning A** 25: 757-60

Fischer M M, Gopal S, Staufer P, Steinnocher K 1994 Evaluation of neural pattern classifiers for a remote sensing application. Paper presented at the 34th European Congress of the Regional Science Association, Groningen, August 1994

Goldberg D E 1989 **Genetic algorithms in search, optimization and machine learning**. Addison Wesley, Reading

Goodchild M F, Haining R P, Wise S et al 1992 Integrating GIS and spatial analysis: Problems and possibilities. **International Journal of Geographical Information Systems** 6 (5): 407-23

Gopal S, Fischer M M 1996 Learning in single hidden-layer feedforward network models. **Geographical Analysis** 28 (1): 38-55

Gopal S, Fischer M M 1993 Neural net based interregional telephone traffic models. In **Proceedings of the International Joint Conference on Neural Networks 2**. IEEE Press, Nagoya: 2041-4

Griffith D A 1993 Which spatial statistics techniques should be converted to GIS functions. In Fischer M M, Nijkamp P (eds) **Geographic information systems, spatial modelling, and policy evaluation**. Springer, Berlin: 101-14

Haining R P 1994 Designing spatial data analysis modules for geographical information systems. In Fotheringham S, Rogerson P (eds) **Spatial analysis and GIS**. Taylor & Francis, London: 45-63

Hajek B 1988 Cooling schedules for optimal annealing, **Mathematics of Operations Research** 13: 311-29

Hecht-Nielsen R 1990 **Neurocomputing**. Addison-Wesley, Reading (Ma.)

Hertz J, Krogh A, Palmer R G 1991 **Introduction to the theory of neural computation**. Addison-Wesley, Redwood City (CA)

Hornik K, Stinchcombe M, White H 1989 Multilayer feedforward networks are universal approximators. **Neural Networks** 2: 359-66

Holland J H 1975 **Adaptation in natural and artificial systems**. University of Michigan Press, Ann Arbor

Jacobs R A 1988 Increased rates of convergence through learning rate adaptation. **Neural Networks** 1: 295-307

Kirkpatrick S, Gelatt Jr C D, Vecchi M P 1983 Optimization by simulated annealing. **Science** 220: 671-80

Kosko B 1992 **Neural networks and fuzzy systems**. Prentice Hall, Englewood Cliffs [NJ]

Koza J R 1994 **Genetic programming II**. The MIT Press, Cambridge (CA)

Koza J R 1992 **Genetic programming**. The MIT Press, Cambridge (CA)

Langton C G (ed) 1989 **Artificial life**. Addison-Wesley Publishing, Reading.

Leung Y 1996 Feedforward neural network models for spatial pattern classification. In Fischer M M, Getis A (eds) **Recent developments in spatial analysis**. Springer, Berlin (in preparation)

Leung Y, Leung K S, Fischer M M, Ng W, Lau M K 1996 The evolution of multilayer feedforward neural networks for spatial interaction using genetic algorithms. Working Paper, Department of Geography, Chinese University of Hongkong

Lindsey C S, Lindblad T 1995 Survey of neural network hardware. **Proceedings of Applications and Science of Artificial Neural**, Networks Conference, SPIE Vol.. 2492, Part Two: 1194-205.

Nordstrom T, Svensson B 1992 Using and designing massively parallel computers for artificial neural networks. **Journal for Parallel and Distributed Computing** 14(3): 260-85

Openshaw S 1994: Computational human geography: Exploring the geocyberspace, **Leeds Review** 37: 201-20.

Openshaw S 1993a Some suggestions concerning the development of artificial intelligence tools for spatial modelling and analysis in GIS. In Fischer M M, Nijkamp P (eds) **Geographic information systems, spatial modelling, and policy evaluation**. Springer, Berlin: 17-33

Openshaw S 1993b Modelling spatial interaction using a neural net. In Fischer M M, Nijkamp P (eds) **Geographic information systems, spatial modelling, and policy evaluation**. Springer, Berlin: 147-64

Press W H, Teukolsky S A, Vetterling W T, Flannery B T 1992 **Numerical Recipes in C**. Cambridge University Press, Cambridge

Rosenberg C R, Blelloch G 1987 An implementation of network learning on the CM. **Proceedings of the Joint Conference on Artificial Intelligence**, Milan: 329-40

Rumelhart D E, Hinton G E, Williams R J 1986 Learning internal representations by error propagations. In Rumelhart D E, McClelland J L (eds): **Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1**. MIT Press, Cambridge (MA): 318-62

Serbedzija N B 1996 Simulating artificial neural networks on parallel architectures. **Computer** 29(3): 56-63

Shang Y, Wah B W 1996 Global optimization for neural network training, **Computer** 29(3): 45-54

Shanno D 1990 Recent advances in numerical techniques for large-scale optimization, in Miller W T (ed) **Neural Networks for Robotics and Control.** MIT Press, Cambridge (CA): 171-8

Staub R, Schwarz D, Schöneburg E 1991 Simulation of backpropagation networks on transputers. **Neurocomputing** 2(5/6): 199-208

Tobler W 1979 Cellular geography. In Gale S, Olsson G (eds) **Philosophy in geography**. Reidel, Dordrecht: 379-86

Wasserman P D 1993: **Advanced methods in neural computing**. Van Nostrand Reinhold, New York

White H 1990 Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings. **Neural Networks** 3: 535-50

Wilkinson G G 1996 Neurocomputing for earth observation - recent developments and future challenges. In Fischer M M, Getis A (eds): **Recent developments in spatial analysis**. Springer, Heidelberg (in preparation)